Listing of All Claims Including Current Amendments

1. (cancelled)

2. (previously presented) The computer language translator according to claim 14 wherein the translator is a bi-directional translator, said analyzer analyzing the second computer language source code to identify the type of data manipulation that the second computer language source code performs, accessing said emulation library and correlating the type of data manipulation the second computer language source code performs to first computer language source code, the correlation being independent of the context in which the second computer language source code is used, said generator generating re-translated first computer language source code that emulates the type of data manipulation the second computer language source code performs.

3. (cancelled)

4. (previously presented) The computer language translator according to claim 14 wherein said generator performs a name adjustment when an incompatibility between the first computer language source code and the second computer language source code occurs during correlation of the type of data manipulation the first computer language source code performs to the second computer language source code to resolve the incompatibility, the name adjustment being independent of the context in which the incompatibility is located.

5. (previously presented) The computer language translator according to claim 14 wherein the first computer language source code comprises a class.

6. (previously presented) The computer language translator according to claim 5 wherein the class consists of: methods, data fields, inner-classes, and combination thereof.

7. (previously presented) The computer language translator according to claim 14 wherein the first computer language source code comprises an identifier.

8. (previously presented) The computer language translator according to claim 7 wherein said generator performs a name adjustment when an incompatibility between the identifier and the second computer language source code occurs during correlation of the type of data manipulation the identifier performs to the second computer language source code to resolve the incompatibility, the name adjustment being independent of the context in which the identifier is used.

9. (previously presented) The computer language translator according to claim 14 wherein said generator generates a tagged element inserted in the second computer language source code indicative of a type of data manipulation the first computer language source code performs.

10. (original) The computer language translator according to claim 9 wherein the tagged element comprises information selected from the group consisting of: formatting, translation data, and first computer language source code.

11. (previously presented) The computer language translator according to claim 14 wherein the first computer language is Java and the second computer language is C++.

12. (previously presented) The computer language translator according to claim 14 wherein the first computer language is Java and the second computer language is C#.

13. (previously presented) The computer language translator according to claim 14 wherein the first computer language is C# and the second computer language is C++.

14. (currently amended) A computer language translating system for translating a first OOP computer language source code to a second OOP computer language source code, comprising:

a computer having a storage;

an emulated Application Programming Interface library having a table and accessible by said computer, said library including data indicative of types of data manipulations between the first computer language source code and the second computer language source code, said table including second computer language equivalent functions callable by said second computer language;

an analyzer analyzing the type of data manipulation the first computer language source code performs, said analyzer accessing said table of said emulation library and correlating the type of data manipulation the first computer language source code performs to a second computer language equivalent function; and

a generator generating second computer language source code based on the identified equivalent functions such that second computer language source code emulates the type of data manipulation the first computer language source code performs;

said second computer language source code providing discreet functionality that is independent from the first computer language source code such that said generated second computer language source code can independently provide the equivalent type of data manipulations provided in said first computer language source code without reference to the first computer language source code.

15. (previously presented) The computer language translating system according to claim 14 wherein the translating system is a bi-directional translator, and further comprises a parser parsing the first and second computer language source codes into parsed elements.

16.-24. (cancelled)

25. (currently amended) A method for translating a first OOP computer language source code to a second OOP computer language source code comprising the steps of:

inputting the first computer language source code into a computer;

parsing the first computer language source code into parsed elements;

analyzing the parsed elements to determine a type of data manipulation the first computer language source code performs;

referencing an emulation Application Programming Interface library including second computer language equivalent functions that emulate the first computer language data manipulations;

selecting second computer language equivalent functions from the emulation library that emulate the first computer language data manipulations;

building class declarations and class definitions from the parsed elements that independently provide the equivalent type of data manipulations provided in the first computer language source code without reference to the first computer language source code;

generating the second computer language source code according to the class declarations and class definitions, the second computer language source code emulating the type of data manipulation the first computer language source code performs;

the second computer language source code referencing the Application Programming Interface library based on the selected equivalent functions emulating the first computer language.

26. (original) The translation method according to claim 25 further comprising the steps of:

parsing the second computer language source code into parsed elements;

analyzing the parsed elements to determine a type of data manipulation the second computer language source code performs;

building class declarations and class definitions from the parsed elements that are independent of the context of the second computer language source code;

generating the re-translated first computer language source code according to the class declarations and class definitions, the first computer language source code emulating the type of data manipulation the second computer language source code performs.

27. (original) The translation method according to claim 25 further comprising the step of, performing a name adjustment when an incompatibility between the first computer language source code and the second computer language source code occurs during translation, the name adjustment being independent of the context in which the incompatibility is located.

28. (original) The translation method according to claim 25 wherein the first computer language source code comprises an identifier.

29. (original) The translation method according to claim 28 further comprising the step of, performing a name adjustment when an incompatibility between the identifier and the second computer language source code occurs during translation, the name adjustment being independent of the context in which the identifier is located.

30. (original) The translation method according to claim 25 further comprising the step of,

generating a tagged element indicative of the type of data manipulation the first computer language source code performs; and

inserting the tagged element in the second computer language source code.

31. (original) The translation method according to claim 25 wherein the first computer language is Java and the second computer language is C++.

32. (original) The computer language translating system according to claim 25 wherein the first computer language is Java and the second computer language is C#.

33. (original) The translation method according to claim 25 wherein the first computer language is C# and the second computer language is C++.

34.-42. (cancelled)

43. (currently amended) A computer language translating system for translating a Java source code to an OOP language source code, comprising:

a computer having a storage;

an emulation Application Programming Interface library having a table and accessible by said computer including data indicative of types of data manipulations between the Java source code and the OOP language source code, said table including OOP language equivalent functions callable by said OOP computer language;

an analyzer analyzing the Java source code to determine a type of data manipulation the Java source code performs, said analyzer accessing said table of said emulation library and correlating the type of data manipulation the Java source code performs to a OOP language equivalent function; and

a generator generating OOP language source code based on the identified equivalent functions such that OOP language source code emulates the type of data manipulation the Java source code performs;

said OOP language source code providing discreet functionality that is independent from the Java source code such that said generated OOP source code can independently provide the equivalent type of data manipulations provided in said Java source code without reference to Java source code.

44. (previously presented) The computer language translating system according to claim 43 wherein said analyzer analyzes the OOP language source code to determine a type of data manipulation the OOP language source code performs, said analyzer accessing said library to correlate the type of data manipulation the OOP language source code performs to Java source code, the correlation being independent of the context in which the OOP language source code is used, and said generator generates re-translated Java source code that emulates the type of data manipulation the OOP language source code performs.

45. (original) The computer language translating system according to claim 43 wherein said analyzer builds class declarations and class definitions based on the type of data manipulation the Java source code performs and said generator generates OOP language source code based upon the class declarations and class definitions.

46. (cancelled)

47. (currently amended) A method for translating a first OOP computer language source code that performs a data manipulation with a virtual machine to a second OOP computer language source code that performs a data manipulation without use of a virtual machine, comprising the steps of:

analyzing the first OOP computer language source code to determine a type of data manipulation the first OOP computer language source code performs;

referencing an emulation Application Programming Interface library including second OOP computer language equivalent functions that emulate the first OOP computer language data manipulations;

correlating the type of data manipulation of the first OOP computer language source code performs to a second OOP computer language equivalent function;

generating the second OOP computer language source code based on the identified equivalent functions such that the second OOP language source code emulates the type of data manipulation the first OOP computer language source code performs; and

the second OOP computer language referencing the Application Programming Interface library to perform ~~performing~~ a data manipulation with the second OOP computer language source code that emulates the type of data manipulation the first OOP computer language source code performs, the second OOP computer language source code performing the data manipulation without use of a virtual machine.

48. (currently amended) A method for translating a first OOP computer language source code that performs a data manipulation with a garbage collector to a second OOP computer language source code that performs a data manipulation without use of a garbage collector, comprising the steps of:

analyzing the first OOP computer language source code to determine a type of data manipulation the first OOP computer language source code performs;

referencing an emulation Application Programming Interface library including second OOP computer language equivalent functions that emulate the first OOP computer language data manipulations;

correlating the type of data manipulation of the first OOP computer language source code performs to a second OOP computer language equivalent function;

generating the second OOP computer language source code based on the identified equivalent functions such that the second OOP language source code emulates the type of data manipulation the first OOP computer language source code performs; and

the second OOP computer language referencing the Application Programming Interface library to perform ~~performing~~ a data manipulation with the second OOP computer language source code that emulates the type of data manipulation the first OOP

computer language source code performs, the second OOP computer language source code performing the data manipulation without use of a garbage collector.